JavaScript voor beginners (deel 1)

Pagina's met pit

Je hebt je eerste pasjes gewaagd in webdesign, en die pagina's op je webstek mogen er heus wel wezen. Alleen vind je ze nogal tam, en van enige interactiviteit met je bezoekers is al helemáál geen sprake! Met enkele eenvoudige javascripts breng je daar zó verandering in... TOON VAN DAELE

Misschien heb je voor die eerste webpagina's wel dankbaar gebruik gemaakt van een zogenoemde WYSIWYG-editor. Dat is een programma waarmee je een webstek in elkaar kan steken zonder één regeltje code te doorworstelen. What you see is what you get, luidt het devies: je hoeft enkel nog de bouwstenen aan te slepen en je ziet onmiddellijk het (grafische) resultaat, net zoals je die webpagina in je browser zou bekijken. Maar vergis je niet: onderliggend bevat elke webpagina wel degelijk de nodige coderegels, *html* (HyperText Markup Language) genoemd. Je begrijpt meteen wat we bedoelen als je in je browser een willekeurige webpagina opent en in het menu **Beeld** de optie **Broncode** of **Paginabron** aanklikt. Het grootste manco aan html is dat deze taal nauwelijks enige dynamiek of interactiviteit in de webpagina's weet te stoppen. *JavaScript* (momenteel aan versie 1.6 toe) is echter de geknipte aanvulling! Het gaat om een vrij eenvoudige programmeertaal die je eigenlijk net zo goed in je kladblok kan intikken. Verwar JavaScript overigens niet met Java: dát is pas een programmeertaal om U tegen te zeggen. De programmaatjes die je met JavaScript maakt – scripts genoemd – dien je dan in je html-bestanden op te nemen. Hoe je zo'n scripts creëert en hoe je die precies inbedt in je (bestaande) webpagina's, verduidelijken we in deze tweedelige cursus aan de hand van een reeks praktische voorbeelden. Je kan de scriptjes die we in deze cursus gebruiken ook terugvinden op onze site www.clickxmagazine.be (onder de rubriek **Aanvullers**). Als je de scripts zelf overtikt, let er dan op dat je steeds dubbele aanhalingstekens gebruikt, en geen twee enkele – anders zullen je scripts niet werken.

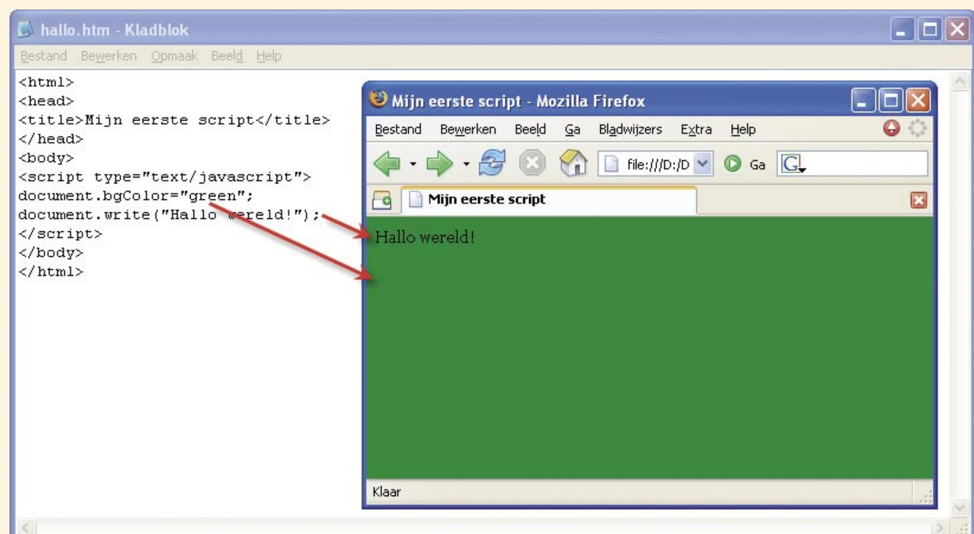
Kladblok (via **Alle Programma's**, **Bureau-accessoires**) en browser in de aanslag? Prima, dan steken we meteen van wal...

Hallo wereld!

We vertelden je al dat je een javascriptje altijd ergens in de html-code van je webpagina moet opnemen. Natuurlijk maak je de browser diets waar dat scriptje precies begint en waar het eindigt. Dat doe je door rond dat script <script>-tags te plaatsen, zoals het volgende voorbeeld duidelijk maakt:

```
<html>
<head>
<title>Mijn eerste script</title>
</head>
<body>
<script type="text/javascript">
document.bgColor="green"
document.write("Hallo wereld!")
</script>
</body>
</html>
```

Tik deze code exact en integraal in je kladblok in en bewaar dit bestand als webpagina, bijvoorbeeld onder de naam hallo.htm. Eventueel kan je de eigenlijke javascript-code ook in een bestaande webpagina kleven. Als je deze pagina vervolgens in je browser opent (via het menu **Bestand**, **Openen**), dan krijg je meteen het resultaat te zien. Veel toelichting behoeft dit simpele scriptje niet, maar het geeft al wel enkele belangrijke basisprincipes van JavaScript aan. Je moet namelijk weten dat JavaScript een objectgebaseerde scripttaal is. Het komt erop neer dat zo'n *script* bestaat uit opdrachten voor objecten.



Zichtbaar resultaat met twee regeltje javascript.

Interactie

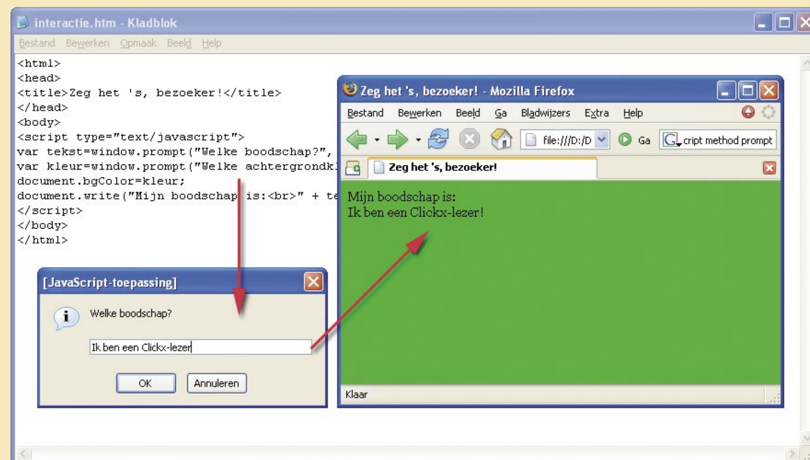
Prima, we zijn op gang gekomen! Alleen... is in ons maiden-script nog helemaal niks te bespeuren van enige interactie met de bezoeker. Daar brengen we snel verandering in: wat zou je ervan denken om onze bezoeker zelf te vragen welke boodschap hij in welke achtergrondkleur te zien wil krijgen? Het volgende script zorgt daarvoor:

```
<script type="text/javascript">
var tekst=window.prompt("Welke boodschap?",➲
"Vervang dit door je eigen tekst")
var kleur=window.prompt("Welke achtergrondkleur➲
(Engels aub)?")
document.bgColor=kleur
document.write("Mijn boodschap is: <br>",➲
tekst)
</script>
```

Een klein script, maar niet zonder nieuwigheden! Je merkt trouwens op dat we zuivere html-regels voortaan achterwege laten: die hoor je natuurlijk zelf nog wel te voorzien.
Beginnen we met het eerste stukje: `var tekst=`. Var staat voor variabele, en dit kan je zien als een container waarin je tijdelijk een of ander stukje gegeven kan onderbrengen. Binnen elk script moet zo'n variabele over een unieke naam beschikken, en deze variabele hebben we tekst gedoopt. Wat we in deze variabele stoppen, lees je achter het gelijkheidsteken: `window.prompt("Welke boodschap?", "Vervang dit door je eigen tekst")`. Hier maken we gebruik van alweer een andere standaardmethode: prompt, en die werkt in op het objectvenster. Aangezien dit het hoofdobject is binnen de hiërarchie

SCRIPTS OVERTIKKEN

Je hoeft de scripts die in deze cursus gebruikt worden helemaal niet over te typen (ze staan immers op onze website onder de rubriek Aanvullers). Ben je toch niet te houden? Hou er dan rekening mee dat sommige regels niet afgebroken mogen worden. Als je op het einde van een coderegel ➲ ziet staan, moet je gewoon blijven verdertikken.



Laat ook de bezoeker z'n zegje doen...

van JavaScript mag je dit object net zo goed onvermeld laten: `var tekst=prompt("Welke...")`. 'Prompt' kunnen we twee argumenten meegeven: het eerste is de mededeling die in het venster verschijnt; het tweede is de tekst die standaard in het tekstveld te zien is. Beide argumenten zijn tekstregels (strings genoemd), en dus dienen we die tussen aanhalingstekens te plaatsen, gescheiden door een komma. Het antwoord dat de bezoeker vervolgens intikt... is precies het gegeven dat door onze instructie in de variabele tekst terecht zal komen. Zolang het script loopt, onthoudt JavaScript dus wat er zich in die variabele bevindt.
Met de volgende scriptregel heb je nu geen moeite meer: die laat zich namelijk precies als de vorige interpreteren. Vervolgens geven we beide variabelen door aan de methodes die we al kennen uit het vorige script. Met de variabele kleur geven we de eigenschap `document.bgColor` euh... kleur, en de methode `document.write` schrijft uit wat de bezoeker had ingetikt. Hoewel, nu ook weer niet helemaal... We hebben deze laatste methode namelijk nog een eigen stukje tekst meegegeven: `"Mijn boodschap is:<br>"`. Wie een beetje html spreekt, herkent in `<br>` (break) meteen de tag waarmee je een nieuwe regel forceert.

Een van de standaardobjecten is 'document', waarmee de op dat ogenblik geopende webpagina wordt bedoeld. Bij zo'n object horen telkens ook een aantal eigenschappen, en in de meeste gevallen kan je (via een script) met een object ook iets doén - een zogenoemde methode.
Grijpen we even terug naar ons voorbeeld. Met de instructie `document.bgColor="green"` zorgen we ervoor dat de achtergrondkleur (background color) van de webpagina groen kleurt. Verder maken we dankbaar gebruik van de ingebouwde schrijfmethode om de wereld gedag te zeggen: `document.write("Hallo wereld!")`. Uiteraard krijg je in deze tweedelige cursus nog heel wat andere objecten, eigenschappen en methodes tussen de kiezen gestouwd. Hou er wel rekening mee dat JavaScript behoorlijk kieskeurig is: een klein tikfoutje - zoals een hoofdletter waar een kleine letter wordt verwacht of een vergeten aanhalingsteken – kan volstaan om je script de mist te doen ingaan. Loopt er inderdaad iets fout, speur dan in eerste instantie naar mogelijke tikfouten!

VAKTAAL A - M N - Z

HTML (HYPERTEXT MARKUP LANGUAGE): De 'programmeertaal' om webpagina's te maken heet html. Bijzonderheid van html is de mogelijkheid om hyperlinks naar andere pagina's te maken. Html-pagina's maken kan je met de hand, of met gespecialiseerde editors als Frontpage of Dreamweaver.

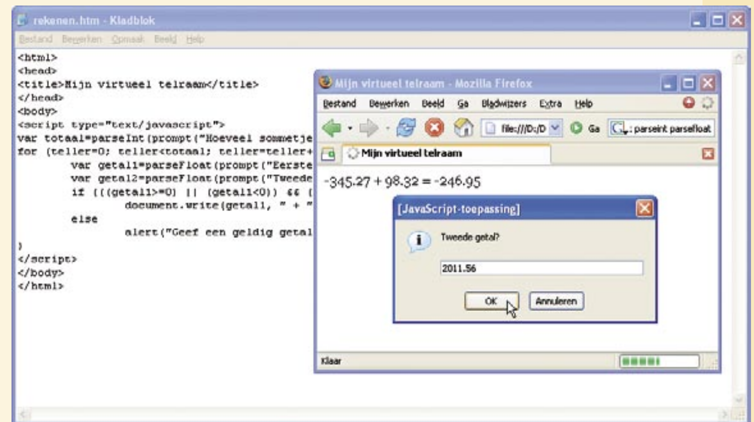
JAVASCRIPT: Eenvoudige scripttaal, losjes gebaseerd op java, waarmee je eenvoudige functies in een webpagina kan programmeren, zoals een klok. Een moderne browser volstaat om javascriptjes te bekijken.

SCRIPT: Lijst van computeropdrachten die in een tekstbestand opgeslagen worden. Een voorbeeld is een loginscript. Scripttalen die veel gebruikt worden zijn VBScript en JavaScript. Ook op het web wordt JavaScript gebruikt.

Reken maar!

Het zal je misschien verbazen, maar JavaScript is ook een ware wiskundige. Het is dus perfect mogelijk je bezoeker een calculator aan te bieden, waarna een scriptje snel de resultaten berekent. Het volgende exemplaar zet je al een flink eind op weg:

```
<script type="text/javascript">
var totaal=parseInt(prompt("Hoeveel ➲
sommetjes aub?"))
for (teller=0; teller<totaal;➲
teller=teller+1) {
      var getal1=parseFloat(prompt("Eerste ➲
getal?"))
      var getal2=parseFloat(prompt("Tweede ➲
getal?"))
      if (((getal1>=0) || (getal1<0)) && ➲
((getal2>=0) || (getal2<0)))
            document.write(getal1, " + ➲
",getal2, " = ", getal1+getal2, "<br>")
      else
            alert("Geef een geldig getal ➲
op aub!")
}
</script>
```



Op JavaScript kan je rekenen!

```
if(((getal1>=0) || (getal1<0))
&& ((getal2>=0) || (getal2<0)))
```

Nog niet echt foolproof, maar toch...

Dit ziet er knap ingewikkeld uit, maar stap voor stap komen we er wel uit... Concentreer je eerst énkel op de scriptregels die we in het groen hebben afgedrukt! Dankzij onze twee vorige scripts weet je die zó te interpreteren, al struikel je wellicht nog wel over de nieuwe methode parseFloat. Die doet niks anders dan de tekstinvoer (namelijk het gegeven dat de gebruiker in het prompt-venster heeft ingetikt) omzetten in een getal, meer bepaald in een getal dat ook cijfers achter de komma mag bevatten. Met de instructie op de laatste groene regel rijgen we dan alles netjes aan elkaar. Merk nogmaals op dat we zuivere tekststrings hier telkens tussen aanhalingstekens plaatsen, maar dat we dat niet doen voor variabelen.

Over nu naar de oranje scriptregels. Die zorgen ervoor dat de bezoeker zelf kan bepalen hoeveel rekensommetjes hij uitgevoerd wil zien. Het antwoord op die vraag wordt dan in de variabele totaal gegoten, maar de methode parseInt zorgt er wel eerst voor dat de invoer van de bezoeker netjes in een geheel getal wordt omgezet. In de tweede oranje regel treffen we een compact stukje code aan. Het gaat hier om een lusinstructie, die ervoor zorgt dat alles wat zich tussen { en } bevindt, een aantal keer wordt uitgevoerd. Deze lusinstructie bestaat uit drie vaste onderdelen: eerst zetten we de teller op 0, wat we doen `via teller=0` (deze variabele hebben we hier dus teller genoemd). Vervolgens leggen we de voorwaarde vast waaraan voldaan moet zijn om de coderegels tussen { en } uit te voeren: `teller<totaal`. Stel dat de bezoeker 3 rekensommetjes wou en dat het de eerste keer is dat de lusinstructie wordt uitgevoerd. In dat geval zal de teller nog op 0 staan, en het totaal op 3. Aan de voorwaarde (teller moet kleiner zijn dan 3) is dus voldaan, en de instructies tussen { en } zullen effectief worden uitgevoerd. Vooraleer dat gebeurt, wordt echter éérst het derde onderdeel van de lus uitgevoerd: `teller= teller +1`. Anders gezegd: de huidige inhoud van de variabele teller wordt met de waarde 1 verhoogd, zodat de teller momenteel op 1 komt te staan (0+1=1). Je hoeft geen bolleboos te zijn om in te zien dat de lus op deze manier precies evenveel keer wordt doorlopen als het getal dat de bezoeker had ingetikt.

Eigenlijk had je net zo goed met een variant op deze lusinstructie kunnen werken. Het script zou er dan als volgt uitzien:

```
var totaal=parseInt(prompt("Hoeveel ➲
sommetjes aub?"))
var teller=0
while (teller<totaal) {
//hier komt de rest van de instructies, ➲
te beginnen met var getal1=…
teller=teller+1
}
```

Resten ons nog de blauwe coderegels. Daarmee bouwen we een rudimentaire controle in op de invoer van de bezoeker. Er wordt hem namelijk gevraagd een getal in te tikken, maar als hij per ongeluk een letter ingeeft, dreigt hij op een scriptfout vast te lopen. En dat vermijden we liever. De eerste blauwe regel laat zich eigenlijk als volgt interpreteren: als variabele getal1 groter dan of gelijk is aan 0 OF ALS (javascript-symbool: || ; dit teken krijg je door de toetscombinatie **AltGr + 1** in te tikken) die kleiner is dan 0 - met andere woorden: als het een geldig getal is – EN ALS (javascript-symbool: &&) datzelfde ook geldt voor getal2, dan mag je de instructie uitvoeren die er onmiddellijk op volgt. In het andere geval voer je de instructie uit die volgt op else. Deze laatste maakt gebruik van de alert-methode, de tweelingbroer zeg maar van de prompt-methode. Belangrijkste verschil tussen beide: de bezoeker krijgt bij alert enkel een waarschuwend vingertje te zien, en kan zelf niks intikken in het venster.

Dag, bezoeker!

Het vorige script was behoorlijk zwaar, maar dat maken we meteen goed door je een lichter verteerbaar exemplaar voor te schotelen:

```
<html>
<head>
<title> Dag, bezoeker! </title>
<script type="text/javascript">
function welkom() {
      alert("\nWelkom op mijn site." + "\nIk ➲
wens je een leuke surftocht toe!")
}
</script>
</head>
<body bgColor="yellow" onLoad="welkom()">
Dit is mijn stek...
</body>
</html>
```

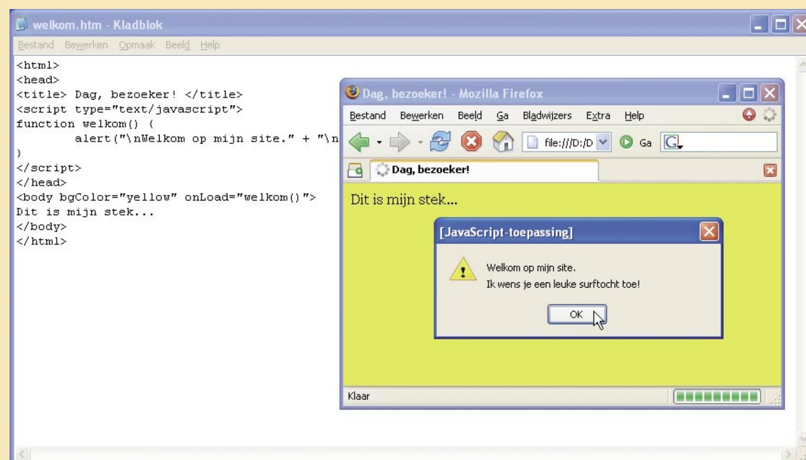
De kern van dit script heb je natuurlijk meteen door: op een zeker ogenblik verschijnt er een apart venstertje, met daarin een welkomstwoord voor de bezoeker. Dit gebeurt meer bepaald meteen bij het inladen van de webpagina, en daar zorgt de instructie <body onLoad="…"> voor. In JavaScript noemen we dit fenomeen een 'event handler'. Een 'event' is een actie die de gebruiker uitvoert terwijl hij je webpagina bezoekt, en via een 'event handler' maak je de browser duidelijk wanneer die actie precies moet plaatsvinden. In dit geval is dat zodra de pagina wordt ingeladen (onLoad). Later in deze cursus zien we nog andere event handlers, zoals onMouseover en onSubmit…

Je vraagt je wellicht wel af wat \n doet in de alert-methode. Heel eenvoudig: dat is de code waarmee we alert verplichten de daaropvolgende tekst op een andere regel te plaatsen – vergelijkbaar dus met
 in html. En misschien vraag je je ook af waarom we die alert-methode niet meteen achter onLoad= hebben geplaatst, en wel als volgt: <body onLoad=" alert('\nWelkom op mijn site.' + '\ nIk wens je een leuke surftocht toe!')">. Dat had inderdaad perfect gekund. Maar let dan wel op het gebruik van aanhalingstekens! Wil je aanhalingstekens nesten binnen andere, dan moet je voor elk genest paar afwisselend enkelvoudige (' ') en dubbele (" ") tekens gebruiken (zie afbeelding).

In dit script hebben we er echter voor geopteerd om zoveel mogelijk javascript-code buiten de eigenlijke <body>-tags van onze webpagina te houden. Dat houdt je pagina's alvast wat overzichtelijker, vooral als je later meer scripts gaat inzetten. Dat kan door de benodigde javascript-regels in een soort vangnet te plaatsen, waarna het volstaat om vanuit de <body>-tags naar dat vangnet te verwijzen. Iets technischer luidt dat: we creëren een functie en doen vanuit de <body>-tags een functie-aanroep. Zoals je merkt, hebben we in dit voorbeeldscript de functie welkom() gecreëerd. De functie zelf plaatsen we (bij voorkeur) tussen de <head>-tags van onze webpagina, en wel als volgt:

```
functie welkom() {
      [de benodigde javascript-code voor deze ➲
functie]
}
```

De aanroep gebeurt dan op de volgende manier: <body onLoad="welkom()">. Werken met functies heeft het bijkomende voordeel dat je de nodige code maar één keer hoeft op te nemen, zelfs als je die vanop dezelfde pagina verschillende keren wil aanroepen.



Een welkomstwoordje, verstopt in een handige functie.

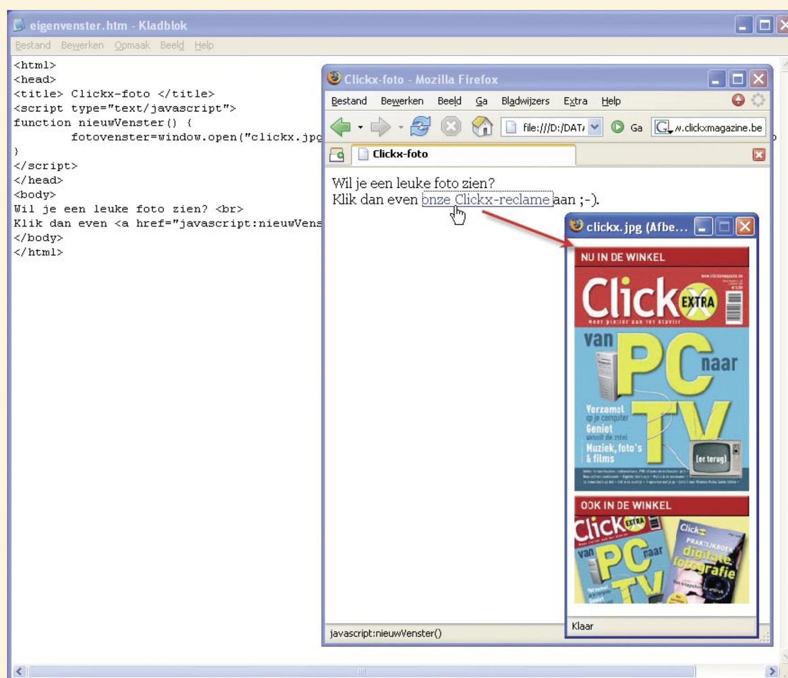
```
<body onLoad="alert('\nWelkom…' + '\nIk…')">
OF
<body onLoad='alert("\nWelkom…" + "\nIk…")'>
```

Verslik je niet in de aanhalingstekens!

Eigen browservenster

Nu je het gebruik van functies een beetje door hebt, zijn we pas goed gelanceerd. Met de volgende functie creëren we een nieuw browservenster naar eigen keuze, en laten daarin op aanvraag van onze bezoeker bijvoorbeeld een leuke foto verschijnen:

```
<html>
<head>
<script type="text/➲
javascript">
function nieuwVenster() {
      fotoVenster=window.➲
open("clickx.jpg","mijnVenster
","width=210,height=420,➲
tool-bars=no,scrollbars=no")
}
</script>
</head>
<body>
Wil je een leuke foto ➲
zien? <br>
Klik dan even ➲
<a href="javascript:➲
nieuwVenster()"> onze ➲
Clickx-reclame </a> aan ;-).
</body>
</html>
```



Creëer je eigen pop-up-vensters.

Ook in dit script hebben we een eigen functie in het leven geroepen. Die luistert naar de naam nieuwVenster() en wordt aangeroepen wanneer de bezoeker de link onze Clickx-reclame aanklikt. Dat gebeurt op de volgende manier: `<a href="javascript:nieuwVenster()"> onze Clickx-reclame </a>`. Bekijken we nu de functie zélf van dichterbij. Zoals je ziet, stoppen we het resultaat van de methode window.open in de variabele met de naam fotoVenster. En wat is nu het resultaat van deze methode? Een nieuw browservenster waarin we meteen de afbeelding clickx.jpg stoppen (hier hadden we net zo goed ook een ganse webpagina kunnen oproepen), dat we de naam mijnVenster hebben gegeven, waarvoor we het gewenste formaat (breedte en hoogte) hebben opgegeven en waarin noch werkbalken noch schuifbalken mogen voorkomen.

Geanimeerde statusbalk

Het kan natuurlijk ook wat speelser. Wat zou je er bijvoorbeeld van denken om de statusbalk van het browservenster in Internet Explorer op te fleuren – of in de ogen van sommige bezoekers wellicht te besmeuren – met een bewegende boodschap? Met één functie krijg je dat voor elkaar, maar het is wel een vrij pittig exemplaar geworden:

```
<html>
<head>
<script type="text/javascript">
teller=0
tekst=" Welkom op mijn homepage. Keer snel ➲
terug, want elke week is er wel iets nieuws!"
function animatie() {
window.status=tekst.substring(teller,tekst.➲
length) + tekst.substring(0,teller-1)
if (teller<tekst.length) {
      teller=teller+1
}
else {
      teller=0
}
setTimeout("animatie()",75)
}
</script>
</head>
<body onLoad="animatie()">
Kijk 's naar je statusbalk ;-)
</body>
</html>
```

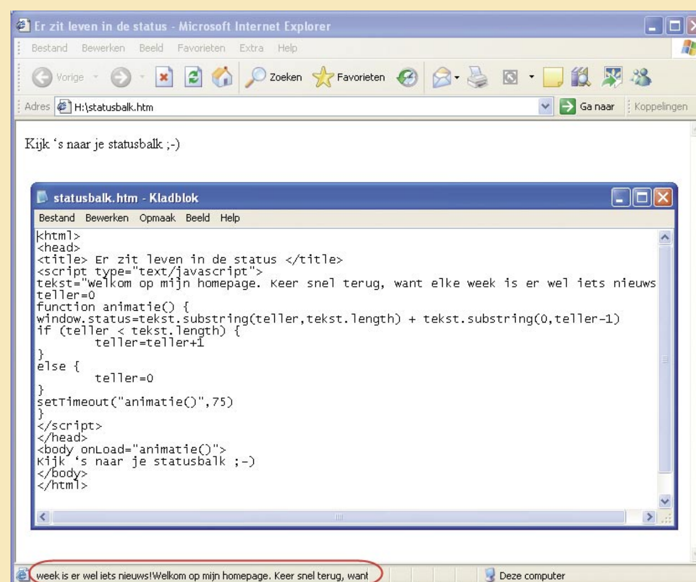
Heel wat onderdelen van dit javascript heb je intussen al min of meer onder de knie. We concentreren ons dus vooral op de nieuwigheden. We stoppen de boodschap die op de statusbalk moet verschijnen eerst in de variabele tekst, en initialiseren alvast onze teller (we zetten die op 0). Straks zie je wel waarom we zo'n tellertje nodig hebben.

De eerste instructie van de functie animatie() doet weinig meer dan het object window.status – dat is Javascripts voor de statusbalk van je browservenster – opvullen met gegevens. Om te begrijpen welke gegevens dat precies zijn, moet je weten hoe de methode substring in elkaar steekt. Deze methode haalt een welbepaald onderdeel uit de ganse tekststring van onze variabele tekst, met name het deel dat zich bevindt tussen de eerste en de tweede parameter van die methode. Passen we deze theorie even toe op het voorbeeld `tekst.substring(teller,tekst.length)`. In dit geval gaat het om het tekstdeel tussen het teken met de waarde teller (die staat initieel op 0, en is het allereerste teken van de string) en het teken dat wordt

aangegeven door de methode `tekst.length`, die het totaal aantal tekens van de tekststring aangeeft (voor onze boodschap zijn dat 80 tekens, inclusief spaties). Concreet geeft de substring(0,80) ons dus alle tekens te zien tussen het allereerste en het allerlaatste teken: de gánse tekststring dus.

We negeren even de rest van deze coderegel en concentreren ons op de if/else-constructie. Daar wordt nagegaan of de teller nog altijd kleiner is dan de lengte van de variabele tekst (80). Is dat inderdaad zo, dan wordt de teller met één verhoogd. In het andere geval wordt de teller opnieuw op 0 gezet. We spitten zo meteen de instructie `set Timeout("animatie()",75)` uit, maar neem voorlopig het volgende van ons aan: die zorgt ervoor dat de functie animatie() telkens opnieuw wordt uitgevoerd. Stel even dat dit inmiddels al vijf keer is gebeurd. In dat geval zal de teller op 4 staan, en wordt de statusbalk met de volgende inhoud opgevuld: tekst.substring(4,80) + tekst.substring(0,3). Dat houdt dus in dat op de statusbalk de volgende tekst te zien is: "om op mijn homepage. Keer snel terug, want elke week is er wel iets nieuws! Welk". Eén ronde verder – de teller staat dan op 5 - wordt dat: "m op mijn homepage. Keer snel terug, want elke week is er wel iets nieuws! Welko". Zo gaat dat maar verder, en dat geeft de optische illusie dat de tekst op de statusbalk zich van rechts naar links beweegt. Na 80 rondjes springt de teller op 0, en kan de carrousel opnieuw beginnen.

Maar hoe zit dat nu precies met de ingebouwde instructie `setTimeout ("animatie()",75)`. Zoals gezegd, roept die telkens opnieuw de functie animatie() op. Hoe frequent dat gebeurt, bepaal je via de tweede parameter (75). Die geeft in milliseconden aan hoelang de instructie moet pauzeren alvorens de functie weer wakker te schudden. Vul je hier het getal 1000 in, dan duurt het welgeteld 1 seconde vooraleer je boodschap zich weer in beweging zet. ♦



Breng wat leven in de statusbalk!

Volgende aflevering: webformulieren, grafische animaties (rollovers), én meer...